Jacobs University Bremen

CO-522-B Communications Basics Lab

Fall 2021

Lab Experiment 4: Digital Phase Locked Loop

Priontu Chowdhury

Introduction

In this las, we designed a simple PLL with a single-pole loop filter, and simulated the response of the PLL in MATLAB. The block diagram below shows the structure of the basic PLL that we implemented:

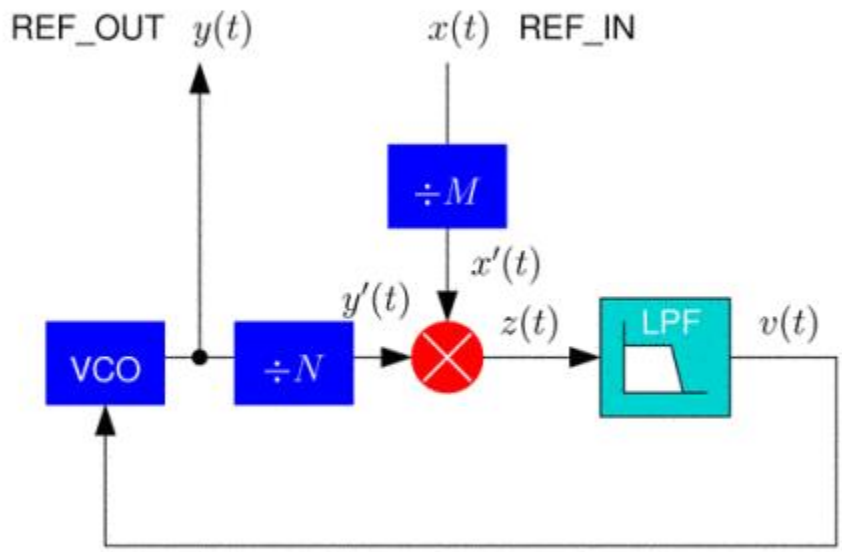

Figure 1: Basic PLL Structure

We take an incoming reference sinusoid and generate an output sinusoid whose phase closely tracks the phase of the input. Using PLL, we can generate a signal that is a specified rational multiple of the input reference frequency or shift the phase by a prescribed amount.

Execution

Simple MATLAB Implementation

The source code for a simple implementation of PLL is provided below:

pll_init.m

```
function [s]=init_PLL(f, D, k, w0, T, table_size)
%parameter
%f - nominal ref frequency
%D - Damping factor
%k - loop gain
%w0 - loop corner frequency
 s.f = f;
 s.D = D;
 s.k = k;
 s.w0 = w0;
 s.T = T;
%create lookup table
for i=0: table_size -1
 s.sine_table(i+1)= sin(2*pi*i/table_size);
end
```

```
%filer coefficients
s.tau1 = s.k/(s.w0)^2;
s.tau2 = 2*s.D/s.w0 - 1/s.k;
s.a1= -(s.T-2*s.tau1)/(s.T+2*s.tau1);
s.b0= (s.T+2*s.tau2)/(s.T+2*s.tau1);
s.b1= (s.T-2*s.tau2)/(s.T+2*s.tau1);
%Create state variables: initial
s.out_old=0.0;
s.z_old=0.0;
s.v_old=0.0;
s.accum=0.0;
end
```

pll.m

```
function [out, state_out] = PLL(in, N, s)
out = zeros(size(in));
for i = 1:N;
 %We are computing phase difference
 z = in(i)*s.out_old;
 v = s.a1*s.v_old + s.b0*z + s.b1*s.z_old;
 s.accum = s.accum + s.f-(s.k/(2*pi))*v;
 s.accum = s.accum - floor(s.accum);
 %Use sine table to calculate output
 out(i) = s.sine_table(floor(1024*s.accum)+1);
 %update state variables
 s.out_old = out(i);
 s.z_old = z;
 s.v_old = v;
end
state_out=s;
```

pll_test.m

```
[s]=PLL_init(0.1,1,1,2*pi/100,1,1024);
Nb=10; %number of blocks
Ns=100; %number of samples
load('ref_800hz');
in= reshape(ref_in,Ns,Nb);
out = zeros(Ns,Nb);
for n=1:Nb
 [out(:,n),s]=PLL(in(:,n),Ns,s);
 plot(1:length(in(:,n)),in(:,n),1:length(in(:,n)),out(:,n))
 pause
end
y_output= reshape(out,Ns*Nb,1);
y_input=ref_in;
plot(1:length(y_input),y_input);
hold on;
plot(1:length(y_output),y_output,'r')
hold off;
```

simulation results:

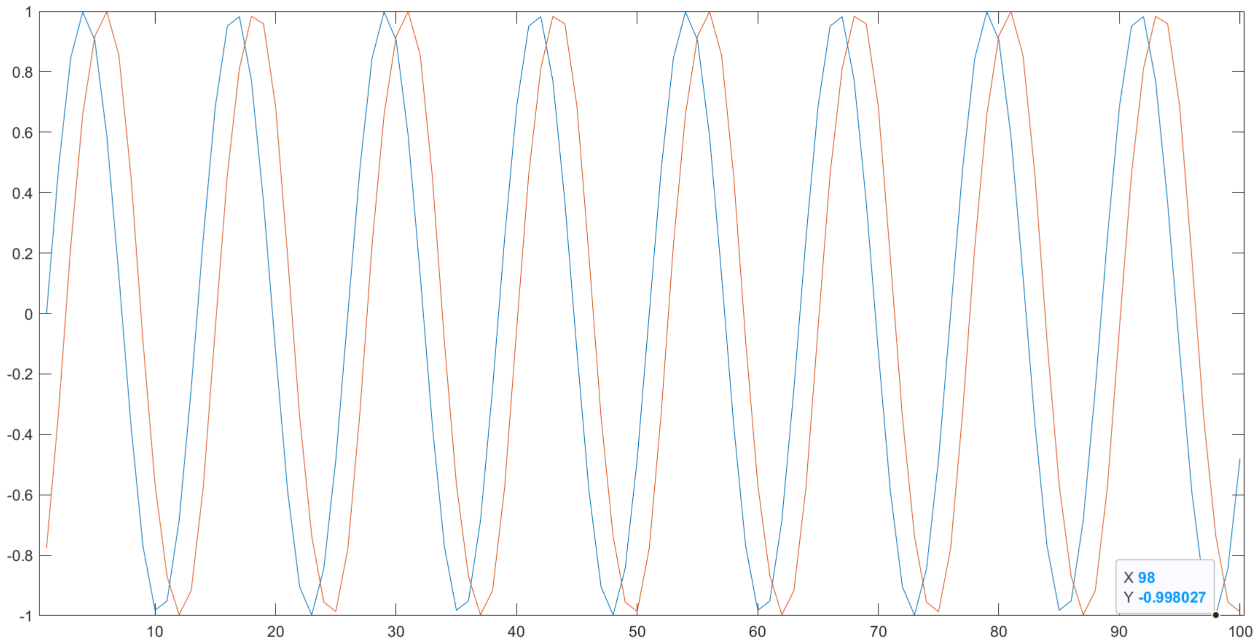

Figure 2: Simulation results

pll_test2.m

```
[s]=PLL_init(0.1,1,1,2*pi/100,1,1024);
Nb=10; %number of blocks
Ns=100; %number of samples
load('ref_stepf');
in= reshape(ref_in,Ns,Nb);
out = zeros(Ns,Nb);
for n=1:Nb
  [out(:,n),s]=PLL(in(:,n),Ns,s);
 plot(1:length(in(:,n)),in(:,n),1:length(in(:,n)),out(:,n))
 pause
end
y_output= reshape(out,Ns*Nb,1);
y_input=ref_in;
plot(1:length(y_input),y_input);
hold on;
plot(1:length(y_output),y_output,'r')
hold off;
```

simulation results:

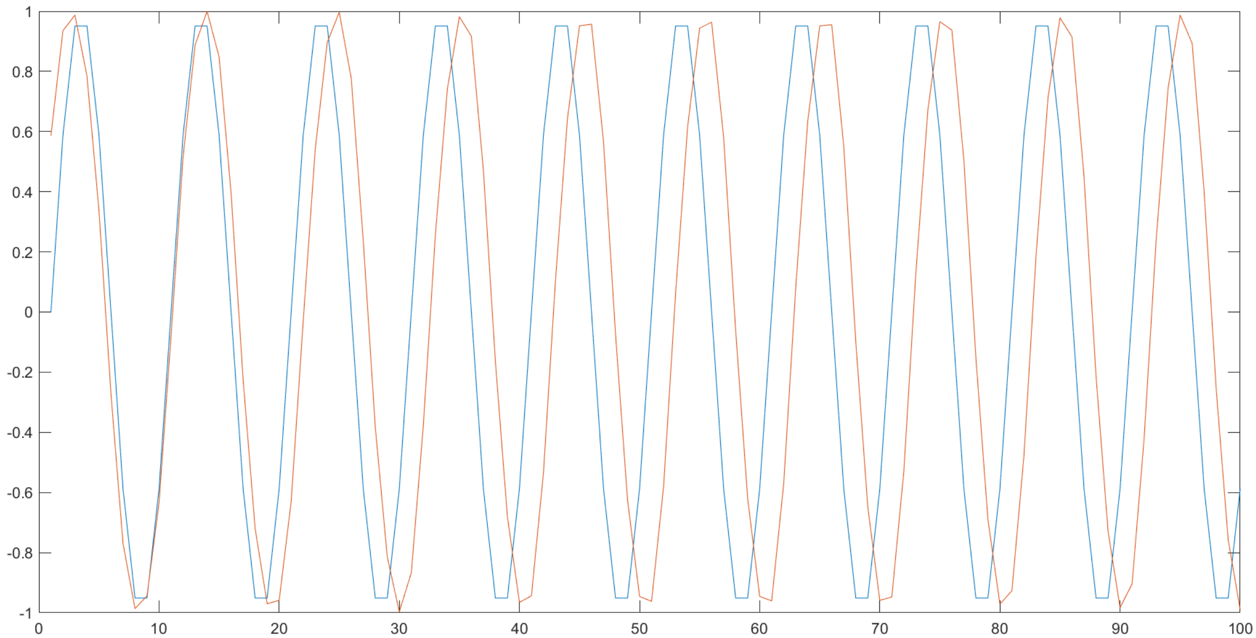

Figure 3: Simulation results for second test

MATLAB Implementation with Arbitrary Amplitude

Implementation code:

pll_init.m

```
function [s]=init_PLL(f, D, k, w0, T, table_size)
%parameter
%f - nominal ref frequency
%D - Damping factor
%k - loop gain
%w0 - loop corner frequency
 s.f = f;
 s.D = D;
 s.k = k;
 s.w0 = w0;
 s.T = T;
%create lookup table
for i=0: table_size -1
 s.sine_table(i+1)= sin(2*pi*i/table_size);
end
%filter coefficients
s.tau1 = s.k/(s.w0)^2;
s.tau2 = 2*s.D/s.w0 - 1/s.k;
s.a1= -(s.T-2*s.tau1)/(s.T+2*s.tau1);
s.b0= (s.T+2*s.tau2)/(s.T+2*s.tau1);
s.b1= (s.T-2*s.tau2)/(s.T+2*s.tau1);
%Create state variables: initial
s.out_old=0.0;
s.z_old=0.0;
s.v_old=0.0;
```

```
s.accum=0.0;
end
```

pll.m

```
function [out,state_out] = PLL(in, N, state_in)
out = zeros(size(in));
s = state_in;
%unit amplitude
amp=0;
for sample_idx=1:N
 amp=amp+abs(in(sample_idx));
end
amp_est=amp/N/(2/pi);
for i=1:N
 scale(i)=in(i)/amp_est;
%Compute phase difference
 z=scale(i)*s.out_old;
 v=s.a1*s.v_old+s.b0*z + s.b1*s.z_old;
 s.accum=s.accum+s.f-(s.k/(2*pi))*v;
 s.accum=s.accum-floor(s.accum);
%sine table and calculate output
 out(i)=s.sine_table(floor(1024*s.accum)+1);
%update state variable
 s.out_old=out(i);
 s.z_old=z;
 s.v_old=v;
end
state_out=s;
```

test_pll.m

```
[s]=PLL_init(0.1,1,1,2*pi/100,1,1024);
Nb=10; %number of blocks
Ns=100; %number of samples
load('ref_800hz');
for j= 1:1000
 ref_in(j)= ref_in(j)*3;
end
in_scale = reshape(ref_in,Ns,Nb);
out=zeros(Ns,Nb);
for n=1:Nb
 [out(:,n),s]=PLL(in_scale(:,n),Ns,s);
 plot(1:length(in_scale(:,n)),in_scale(:,n),1:length(in_scale(:,n)),out(:,n))

end
y_output= reshape(out,Ns*Nb,1);
y_input=ref_in;
figure
plot(1:length(y_input), y_input);
hold on;
plot(1:length(y_output) , y_output, 'r');
hold off;
```

Simulation results:

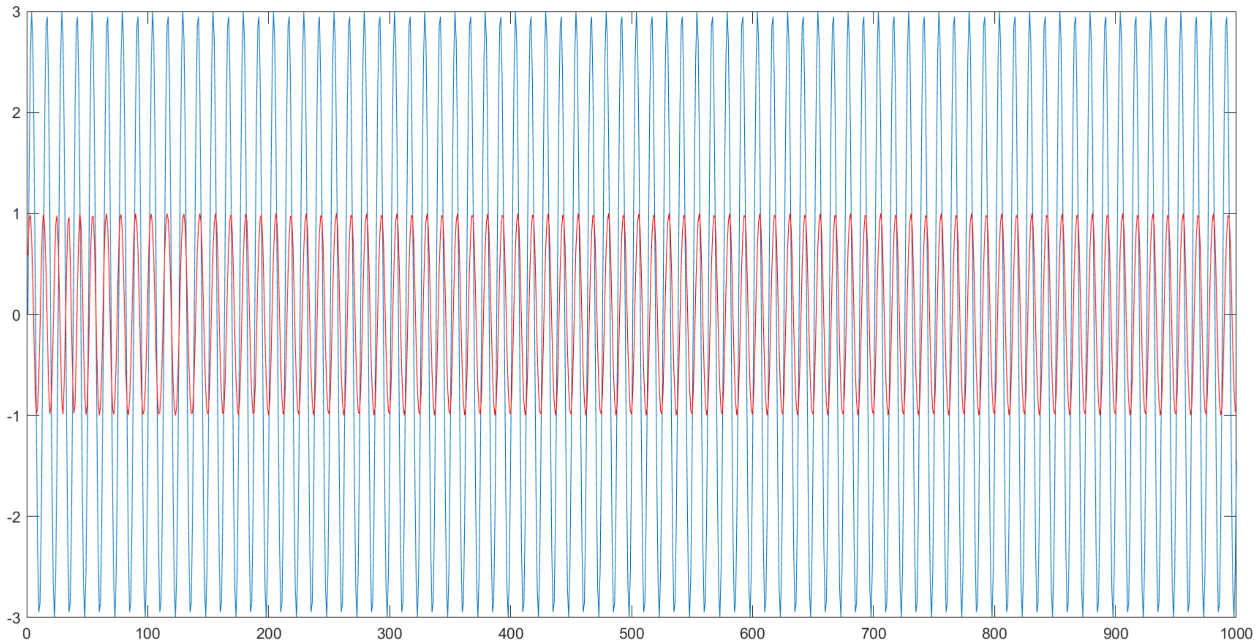

Figure 4: Simulation results 1

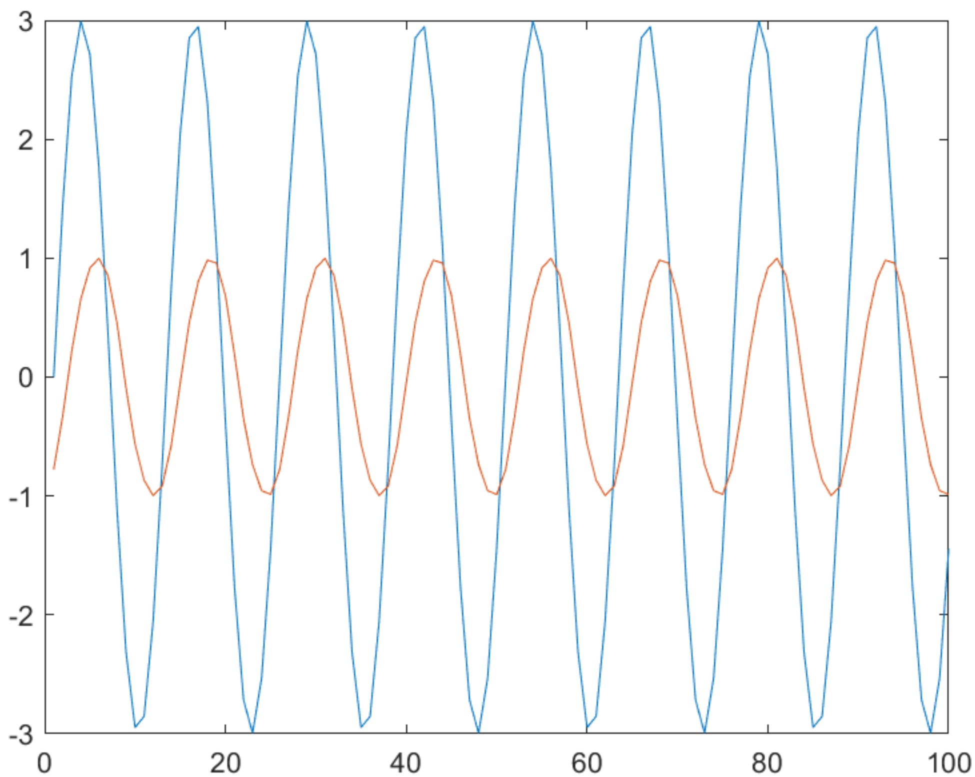

Figure 5: Simulation results 2

Lab Write-Up

<u>Question 1</u>

What are the two functions of a PLL in a communications system?

In communication systems, PLL is used for:

1. Carrier Recovery, where it synchronizes the local oscillator to the incoming signal.
2. Symbol Timing Recovery, which involves properly aligning the sample times at the matched filter output.

2. What are the key components (operations) used to implement a PLL?

The key components are the multiplier, the low pass filter and the VCO. The multiplier takes the product of the incoming signal (input reference) and the VCO (local reference), which results in a low frequency and a high frequency component. The low pass filter removes the high frequency component, leaving the slowly varying voltage which is proportional to the phase difference between the input and reference signals. This phase difference is fed into the VCO. The VCO (voltage controlled oscillator) controls the speed of the oscillator: it speeds up when the output is lagging input and slows down when the output is leading input.

3. When we write the second-order frequency response of the PLL, this expression relates the input and output <u>phases</u> of the references.

4. What does the corner frequency of the PLL loop filter control?

The corner frequency of the PLL controls how fast the loop adapts to the phase changes.

5. How should the loop filter corner frequency compare to the input reference frequency?

The corner frequency is usually lower (less than 0.1 times) than the frequency of the sinusoid or clock being tracked (input reference frequency).

6. What does it mean for a PLL to track?

Tracking means that the PLL is maintaining synchronization between the input and output signal. This entails that the input and the reference signal generated by the VCO have reached proper alignment. Hence, the control voltage goes to 0.

7. What does it mean for the PLL loop to be overdamped or underdamped?

An overdamped PLL takes a long time to adapt to phase changes. An underdamped PLL responds fast to phase changes but tends to overshoot at the target.

8. Why is an accumulator useful for a PLL implementation?

The accumulator is a single real number that keeps track of the current phase. At each time step, the accumulator content specifies the current position in the sine lookup table, where the table holds one sinusoidal cycle. This makes it useful in PLL implementation.

9. Why do we need to sample sin() in our lookup table more finely than at the normal sample rate of the system?

We need to sample more finely in our lookup table because the PLL requires the values which are in between the samples spaced by the sample period.

10.How do we keep our accumulator in the range [0,1]?

We use the following operation:

$$accum = accum - floor(accum)$$

This operation allows the accumulator to wrap when its value increments past 1 or drops below 0.

11.What is the point of storing and restoring the state of the PLL for each block?

We store and restore the state of the PLL for each block because we want to store the current estimate values of phase and amplitude.

12. How do we handle signals with arbitrary amplitude?

For signals with arbitrary amplitude, we average the magnitude of the samples to create an estimate of the amplitude of each block. Consequently, the PLL is modified so that each block of samples is scaled to have approximately unit amplitude. We can then use the estimate to scale te next block of samples.

<u>Question 2</u>

A paper design of your PLL, showing the important parameters that are needed for implementation

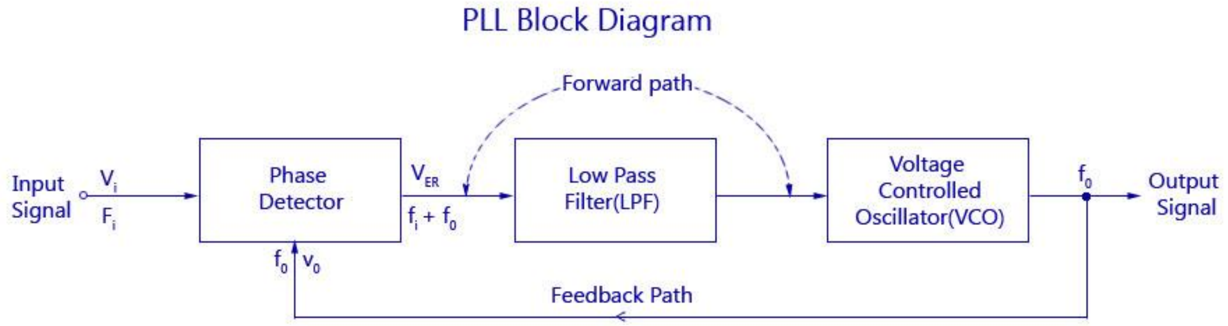

Figure 6: Block diagram of PLL

<u>Question 3</u>

A printout of your final MATLAB implementation of the PLL

Implementation code:

pll_init.m

```
function [s]=init_PLL(f, D, k, w0, T, table_size)
%parameter
%f - nominal ref frequency
%D - Damping factor
%k - loop gain
%w0 - loop corner frequency
 s.f = f;
 s.D = D;
 s.k = k;
 s.w0 = w0;
 s.T = T;
%create lookup table
for i=0: table_size -1
 s.sine_table(i+1)= sin(2*pi*i/table_size);
end
%filter coefficients
s.tau1 = s.k/(s.w0)^2;
s.tau2 = 2*s.D/s.w0 - 1/s.k;
s.a1= -(s.T-2*s.tau1)/(s.T+2*s.tau1);
s.b0= (s.T+2*s.tau2)/(s.T+2*s.tau1);
s.b1= (s.T-2*s.tau2)/(s.T+2*s.tau1);
%Create state variables: initial
s.out_old=0.0;
s.z_old=0.0;
s.v_old=0.0;
s.accum=0.0;
end
```

pll.m

```
function [out,state_out] = PLL(in, N, state_in)
out = zeros(size(in));
s = state_in;
%unit amplitude
amp=0;
for sample_idx=1:N
 amp=amp+abs(in(sample_idx));
end
amp_est=amp/N/(2/pi);
for i=1:N
 scale(i)=in(i)/amp_est;
%Compute phase difference
 z=scale(i)*s.out_old;
 v=s.a1*s.v_old+s.b0*z + s.b1*s.z_old;
 s.accum=s.accum+s.f-(s.k/(2*pi))*v;
 s.accum=s.accum-floor(s.accum);
%sine table and calculate output
 out(i)=s.sine_table(floor(1024*s.accum)+1);
%update state variable
 s.out_old=out(i);
 s.z_old=z;
 s.v_old=v;
end
```

```
state_out=s;
```

test_pll.m

```
[s]=PLL_init(0.1,1,1,2*pi/100,1,1024);
Nb=10; %number of blocks
Ns=100; %number of samples
load('ref_800hz');
for j= 1:1000
 ref_in(j)= ref_in(j)*3;
end
in_scale = reshape(ref_in,Ns,Nb);
out=zeros(Ns,Nb);
for n=1:Nb
 [out(:,n),s]=PLL(in_scale(:,n),Ns,s);
 plot(1:length(in_scale(:,n)),in_scale(:,n),1:length(in_scale(:,n)),out(:,n))

end
y_output= reshape(out,Ns*Nb,1);
y_input=ref_in;
figure
plot(1:length(y_input), y_input);
hold on;
plot(1:length(y_output) , y_output, 'r');
hold off;
```

<u>Question 4</u>

A plot showing the simulated performance of the final PLL, demonstrating that the PLL can adapt to abrupt changes in frequency/phase

Simulation results:

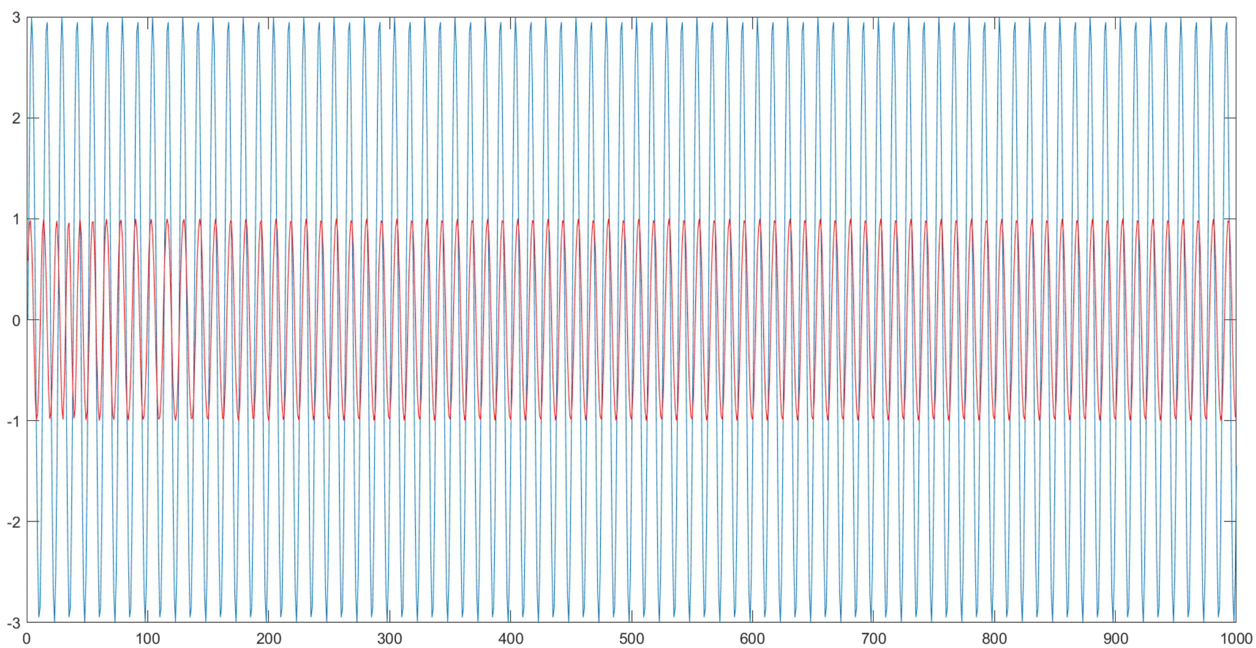

Figure 7: Simulation results 1

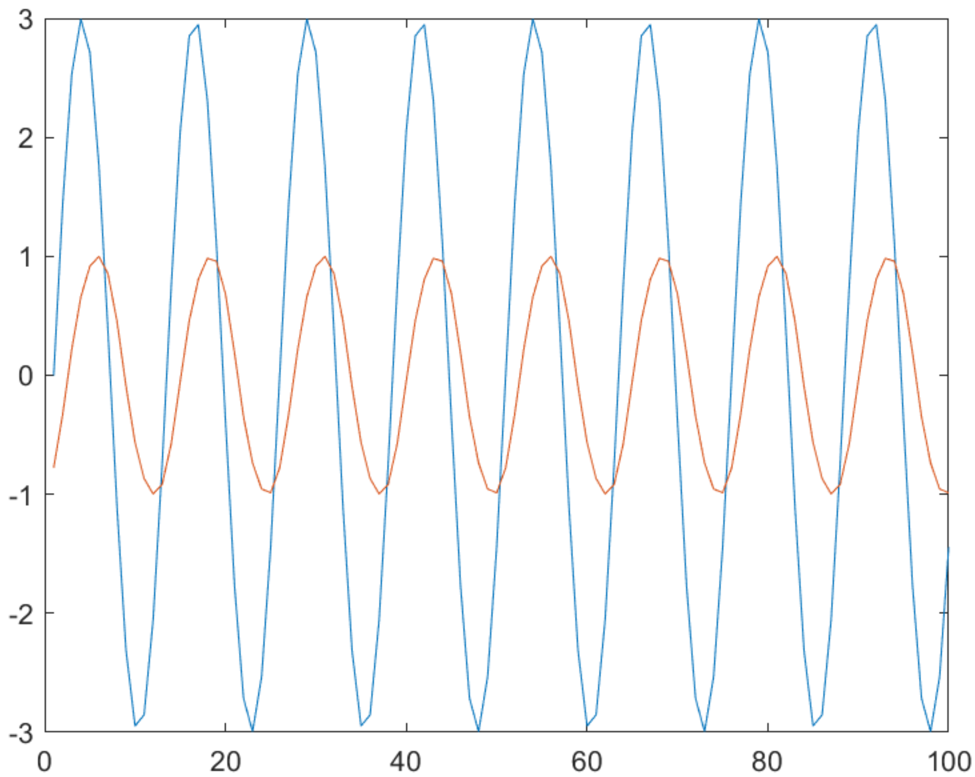

Figure 8: Simulation results 2

Conclusion

In this lab, we worked on Digital Phase Locked Loops, which is designed for tracking the phase of a sinusoidal input in order to achieve synchronization. Synchronization is an important part of signal processing applications. In communications systems, the receiver needs to correctly synchronize its local oscillator and matched filter sample points with respect to the transmitted signal in order to properly decode the information stream. We use phase locked loops in order to accomplish this milestone in communication systems. This lab taught us the mechanism through which this task is accomplished using MATLAB. We were given some datasets on which we run the source code and evaluate the results.

References

Lab Manual: Digital Phase Locked Loop (PLL)